

Authentication for Web APIs

using JSON Web Tokens and Passport

Frank Walsh, 2018

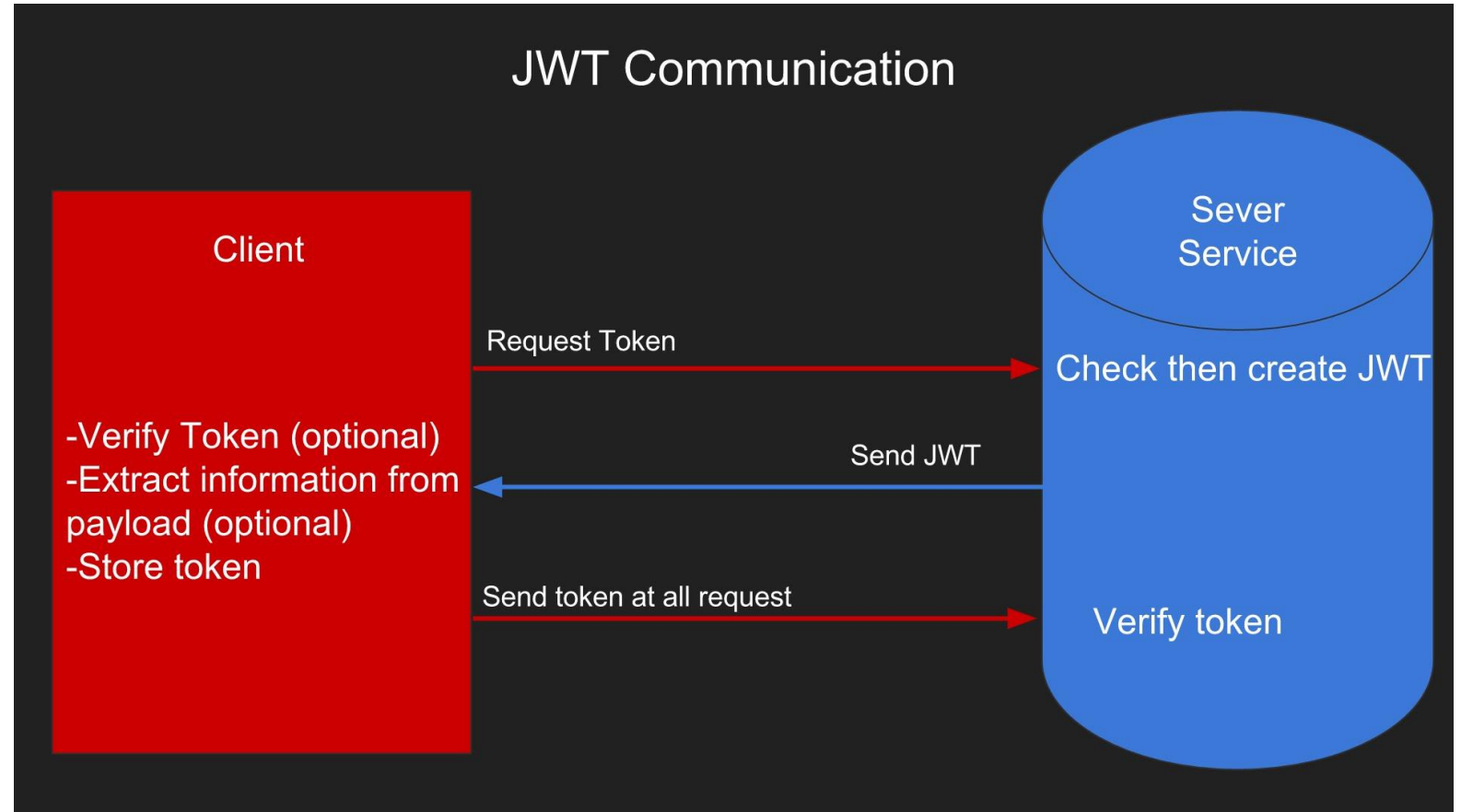
Agenda

- JSON Web Tokens (JWT)
- Authentication
 - Salting with BCrypt
- Passport
- Mongoose Middleware(hooks)
- Use Case – Login/Register using JWT/Passport



Authentication options

- Many solutions for Auth
 - Cookies, basic-auth, JWT, OAuth.
 - Web-based Identity Federation/3rd Party (Firebase)
- JSON Web Tokens (JWT)
 - Tokens means no need to keep sessions or cookies
 - In keeping with REST stateless principle – token sent on each request
 - Token stored on client, usually in local storage of client.



JSON Web Tokens

JWT


[Debugger](#)

[Libraries](#)

[Introduction](#)

[Ask](#)

[Get a T-shirt!](#)

Crafted by  Auth0



JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.

JWT.IO allows you to decode, verify and generate JWT.

[LEARN MORE ABOUT JWT](#)

Username and Password Scenario

- Scenario
 - User signs up to access an API (username & password)
 - Create a new user in database
 - Use new username to create a JWT
 - Send JWT back to user
 - User stores JWT
 - JWT used on every subsequent request to protected resource
- Authentication and Identification
 - ...because username was used to generate JWT.

Authentication Middleware

- Need express middleware to manage user login
- Need Express middleware to restrict access to sensitive routes.
- Options
 - Roll our own
 - Use existing framework/package

```
app.use(function (req, res, next) {  
  if (!userAuthenticated(req)) {  
    return res.redirect('/login');  
  }  
  next();  
});  
  
app.use(express.static(__dirname + '/public'));
```

Passport

- Passport is authentication middleware
- Flexible and modular.
- Easy to retrospectively drop in to an Express app.
- Lots of "strategies" for authentication
 - Username/Password
 - Facebook
 - Twitter





Search for Strategies



15,333

Passport

Simple, unobtrusive authentication for Node.js

Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.



app.js - vim

```
passport.authenticate('github');
```


Passport Overview

- Passport offers different authentication mechanisms as **Strategies**
 - You install just the modules you require for a particular strategy
- Authenticate by calling `passport.authenticate()`
 - specify which strategy to use.
- The **`authenticate()`** function signature is a standard Express middleware function...

Authentication for "Hacker News"



Restrict access to authenticated users.



Provide **User API** to login/register.



Users should only have to
log in once:

Ideally identified and
authenticated in
subsequent requests.



Username and Password authentication.



No clear case passwords
like last week!!!

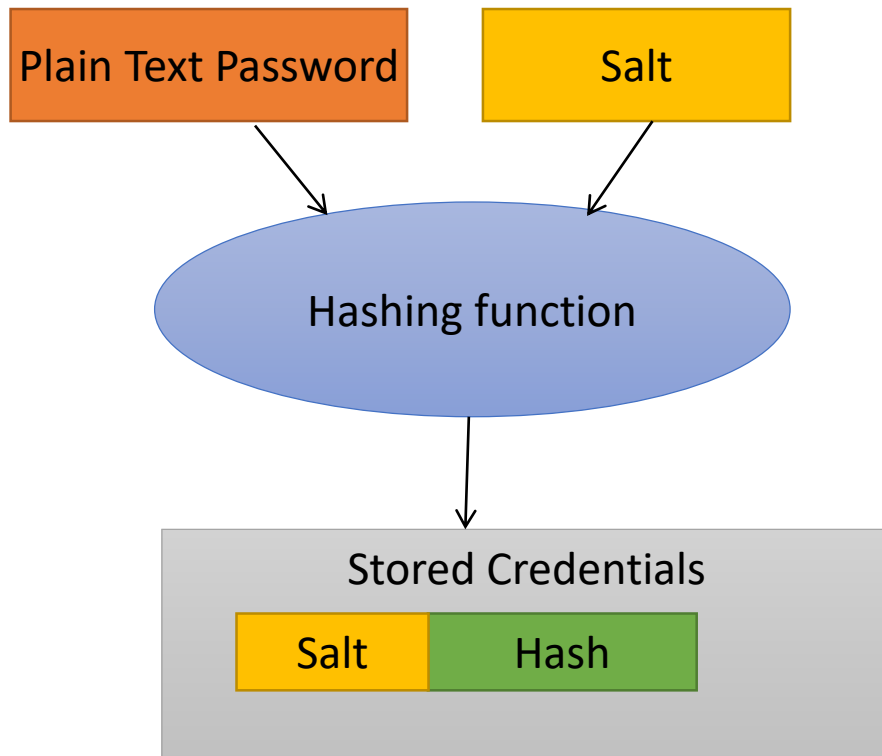
Hash/Salt all passwords in
MongoDB

Web authentication – credentials

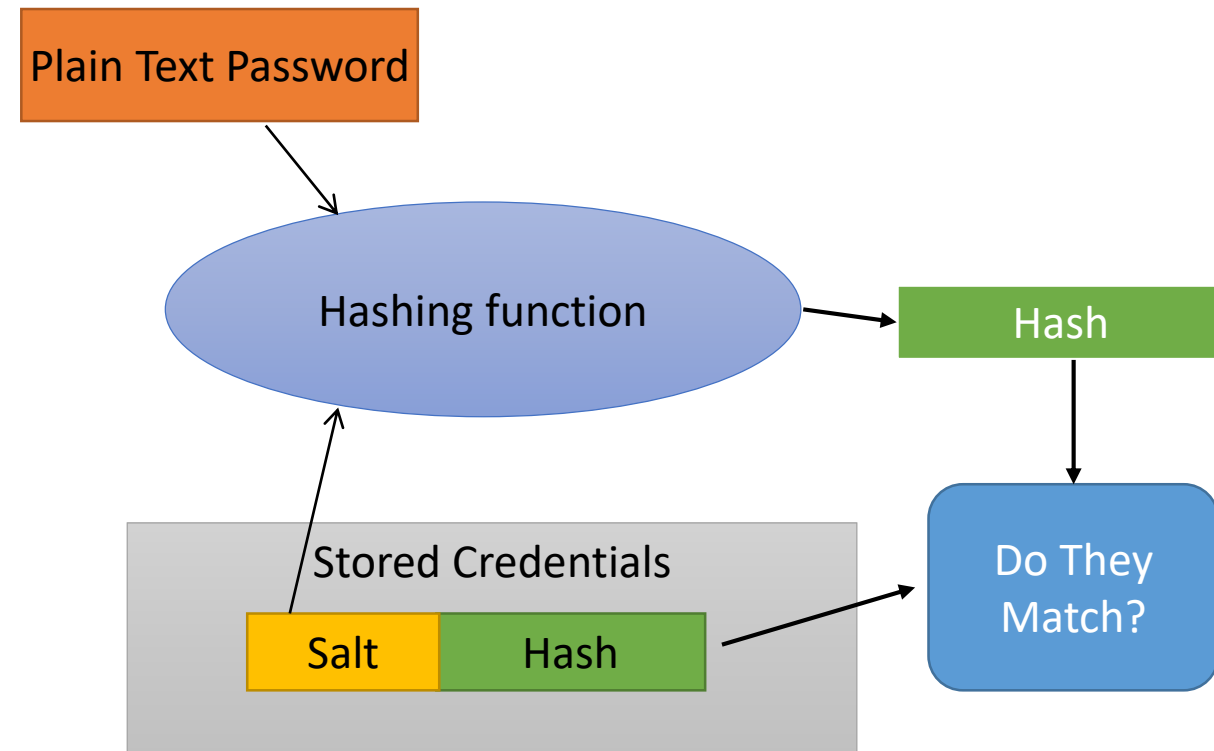
- Credentials should be stored securely in a centralised location
 - Should only be readable by suitably privileged users
 - Credentials should not find their way into hidden fields, headers, cookies
 - Should not be “hard coded”
- Passwords should be “**salted**” and “**hashed**”
 - Salting involves appending random bits to each password
 - Salted password is then hashed (i.e. one-way encrypted) for storage
- Objective is to store something derived from the password that allows an entered candidate password to be checked ...
 - ... but such that the password cannot be retrieved (by *anybody*, even an administrator)

Passwords & Salting

Password Creation



Password Verification



Why Salt?

- Frustrates dictionary attacks.
- Prevents duplicate passwords appearing as duplicates in password file (using different Salts)
- Protects users where same password is reused on different systems/sites.



This Photo by Unknown Author is licensed under [CC BY-SA](#)

Salting in Node.js/Express Apps

bcrypt-nodejs

0.0.3 • Public • Published 6 years ago

Readme

0 Dependencies

744 Dependents

3 Versions

bcrypt-nodejs

Warning: A change was made in v0.0.3 to allow encoding of UTF-8 encoded strings. This causes strings encoded in v0.0.2 or earlier to not work in v0.0.3 anymore.

Native JS implementation of BCrypt for Node. Has the same functionality as `node.bcrypt.js` expect for a few tiny differences. Mainly, it doesn't let you set the seed length for creating the random byte array.

install

```
> npm i bcrypt-nodejs
```

weekly downloads

48,651

version

0.0.3

license

none

- Several NPM packages available.
- Also in other languages (Java)



```
bcrypt.genSalt(10, (err, salt)=> {
  if (err) {
    return next(err);
  }
  bcrypt.hash(user.password, salt, null, (err, hash)=> {
    if (err) {
      return next(err);
    }
    user.password = hash;
    next();
  });
});
```

Mongoose User Model

Create Mongoose User Model

Use Mongoose to specify user
model:

```
import mongoose from 'mongoose';
import bcrypt from 'bcrypt-nodejs';

const Schema = mongoose.Schema;
const UserSchema = new Schema({
  username: {
    type: String,
    unique: true,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
});
```


Mongoose Middleware: Hash/Salt Passwords

- Mongoose supports Middleware (also called pre and post *hooks*).
- Can use, like Express middleware, to process documents
- Use **bcrypt** package to hash and salt passwords

```
UserSchema.pre('save', function(next) {
  const user = this;
  if (user.isModified('password') || user.isNew) {
    bcrypt.genSalt(10, (err, salt)=> {
      if (err) {
        return next(err);
      }
      bcrypt.hash(user.password, salt, null, (err, hash)=> {
        if (err) {
          return next(err);
        }
        user.password = hash;
        next();
      });
    });
  } else {
    return next();
  }
});
```

Mongoose Methods: compare passwords

- You can define instance and static methods in Mongoose Schemas.
- For authentication, define a `comparePassword(..)` instance method
 - Use this to authenticate users
 - **Bcrypt** used to compare with hashed/salted password.

```
UserSchema.methods.comparePassword = function(passw, cb) {  
  bcrypt.compare(passw, this.password, (err, isMatch) => {  
    if (err) {  
      return cb(err);  
    }  
    cb(null, isMatch);  
  });  
};
```

User API: User Routes

- Create new router to support following API

Route	GET	POST	PUT	DELETE
/api/users	List all users	Register/ Authenticate User	N/A	N/A

User API: Get users

```
import express from 'express';
import User from './userModel';
import asyncHandler from 'express-async-handler';
import jwt from 'jsonwebtoken';

const router = express.Router(); // eslint-disable-line

// Get all users
router.get('/', asyncHandler(async (req, res) => {
  const users = await User.find();
  res.status(200).json(users);
}));
```

Create a route to list all users:

User API: Register new user

- Will use query string of URL to indicate action to take on resource
 - **Action===register** will register new user

http://localhost:8080/api/users?action=register

```
// Register/login a user, using async handler
router.post('/', asyncHandler(async (req, res) => {
  if (!req.body.username || !req.body.password) {
    res.json({
      success: false,
      msg: 'Please pass username and password.',
    });
  }
  if (req.query.action === 'register') {
    const newUser = new User({
      username: req.body.username,
      password: req.body.password,
    });
    // save the user
    await newUser.save();
    res.status(201).json({
      success: true,
      msg: 'Successful created new user.',
    });
  }
});
```

User API: Authenticate User

- Find user and compare password using user model
- Generate and return JWT token using username field
- **Client needs to keep token for subsequent messaging**
 - store JWT in local storage.

```
} else {  
  const user = await User.findByUserName(req.body.username);  
  if (!user) return res.status(401).send({success: false, msg: 'Authentication  
  user.comparePassword(req.body.password, (err, isMatch) => {  
    if (isMatch && !err) {  
      // if user is found and password is right create a token  
      const token = jwt.sign(user.username, process.env.secret);  
      // return the information including token as JSON  
      res.status(200).json({  
        success: true,  
        token: 'BEARER ' + token,  
      });  
    } else {  
      res.status(401).send({  
        success: false,  
        msg: 'Authentication failed. Wrong password.',  
      });  
    }  
  });  
};
```

Users API: User Collection

Users Collection	
_id	"5ad46fccada1ab2d67b349ec"
username	"user1"
password	"\$2a\$10\$9r3v12AvPPSkcpJXiohGgehGY50gvgWfV9AAA Bi37rAggsPmxBdwW"
__v	0
1	
_id	"5ad46fccada1ab2d67b349ed"
username	"user2"
password	"\$2a\$10\$YZlmbnUSZhBq9FAsAqKTyOJk8uXEweC7XtTNY/ ozu8aMGXDW07Xxa"
__v	0



Hashed/Salted value for password "test1"

Protecting Routes with Passport

Protecting API Routes: Passport JWT Policy

- Passport strategies are a middleware functions that a requests runs through before getting to the actual route.
- If the authentication strategy fails,
 - callback will be called with an error
 - the route will not be called and a 401 Unauthorized response will be sent.

/auth/index.js

```
import passport from 'passport';
import passportJWT from 'passport-jwt';
import UserModel from '../api/users/userModel';
import dotenv from 'dotenv';

dotenv.config();

const JWTStrategy = passportJWT.Strategy;
const ExtractJWT = passportJWT.ExtractJwt;

let jwtOptions = {};
jwtOptions.jwtFromRequest = ExtractJWT.fromAuthHeaderAsBearerToken();
jwtOptions.secretOrKey = process.env.secret;
const strategy = new JWTStrategy(jwtOptions, async function(payload, next) {
  // usually this would be a database call:
  const user = await UserModel.findByUserName(payload);
  if (user) {
    next(null, user);
  } else {
    next(null, false);
  }
});

passport.use(strategy);

export default passport;
```

Protecting API Routes: initialise and add Middleware

In */index.js* of express app

```
// import passport configured with JWT strategy
```

```
import passport from './auth';
```

```
...
```

```
// initialise passport
```

```
app.use(passport.initialize());
```

```
// Add passport.authenticate(..) to middleware stack for protected routes
```

```
app.use('/api/posts', passport.authenticate('jwt', {  
  session: false  
}), postsRouter);
```

React Apps and JWT

Hacker News App

- Up to now used stubAPI or JSON server
- We want to:
 - Replace with calls to Express HackerNews API
 - Provide login/signin capabilities.
 - Only allow signed in users to see/add posts

Hacker News

👍0 [sfd Comments](#)

👍2 [Coinbase Raises \\$75M from DFJ Growth, USAA, and More Comments](#)

👍10 [India - Tiger population sees 30% increase. Comments](#)

👍12 [Google Nears \\$1B Investment in SpaceX Comments](#)

👍12 [The button that is not. Comments](#)

Add a new post

Proposed Architecture

- Create-React-app uses Webpack development server.
- HackerNews API is an Express.js app.
- Configure Webpack server to "proxy" any unknown requests to Express app
 - Just need "**proxy**":"**http://localhost:8080**" entry in package.json.
- Removes Cross-Origin-Resource-Sharing (CORS) issues with the browser

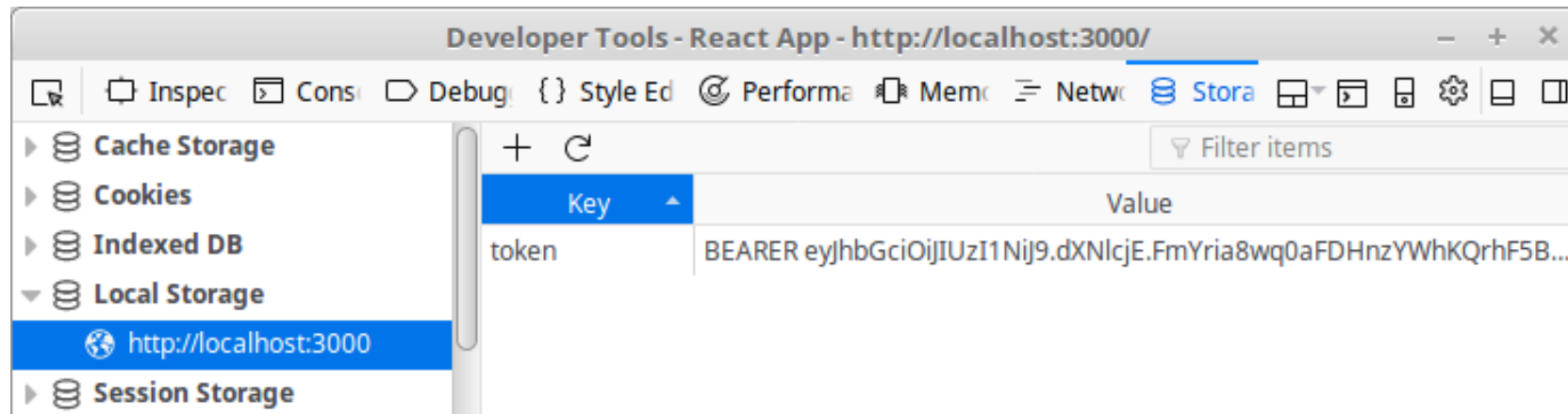


JavaWebToken Storage

- Most browsers/devices have **local storage** .Can access using **localStorage** object.

```
localStorage.setItem('token', token);
```

```
const token = localStorage.getItem('token');
```



Making API HTTP requests

- Use **Axios**, promise-based HTTP client to make requests to the Express API from the React app.
 - `npm install -save axios`

```
import axios from 'axios';
import auth from '../auth';
export const upvote = async (postId) => {
  axios.post(`/api/posts/${postId}/upvote`)
    .then(resp => resp.data);
};

export const getAll = async () => {
  const resp = await axios.get('/api/posts', {headers: {'Authorization': auth.getToken()}});
  return resp.data;
};

export const getPost = async (postId) => {
  const resp = await axios.get(`/api/posts/${postId}`, {headers: {'Authorization': auth.getToken()}});
  return resp.data;
};
```

Integrating with React App

- Use HackerApp component state for posts.
- Use **componentDidMount()** function to the HackerApp component to initialise the posts from the Express API.
 - React Lifecycle Method that runs after all the elements rendered correctly

```
96 class HackerApp extends React.Component {  
97   state = {posts: [{}]};  
98   async componentDidMount () {  
99     try{  
100       const resp = await api.getAll();  
101       this.setState({  
102         posts: resp,  
103         isHidden: false  
104       });  
105     } catch (e){  
106       this.setState({  
107         isHidden: true  
108       });  
109     }  
110   }  
111 };
```

```
133 render() {  
134   const posts = _.sortBy(this.state.posts, post =>  
135     post.upvotes);  
136   return (  
137     <div >  
138       {this.state.isHidden && <Button href="/login">Login/Signup</Bu  
139       {!this.state.isHidden && <NewsList posts={posts}  
140         upvoteHandler={this.incrementUpvote} />}  
141       <Form addHandler={this.addPost} />  
142     </div>  
143   );  
144 }  
145 }
```


Log In Compn

Login/Register Component

- Can try using existing component
 - npm install --save react-signup-login-component
- OR Create **your own loginPage** component
- Add to App router (in index.js)

```
class Login extends Component {
  constructor(props) {
    super(props);

    this.state = {
      username: '',
      password: '',
      confirmPassword: '',
      authenticated: false,
      action: 'Log In'
    };
  }

  handleChange = event => {
    this.setState({
      [event.target.id]: event.target.value
    });
  }

  handleSubmit = event => {
    event.preventDefault();
  }

  signupWasClickedCallback = async (event) => {
    try {
      if (this.state.password !== this.state.confirmpassword) {
        Error('Passwords do not match!');
        alert('Passwords Must Match');
      }
    }
  }
}
```



```
import LoginPage from './components/loginPage';
```

```
<Switch>
  <Route path='/posts/:post_id' component={ CommentPage } />
  <Route path='/login' component={LoginPage} />
  <Route exact path='/' component={ HackerApp } />
  <Redirect from='*' to='/' />
</Switch>
```



Hacker News

Log In

Your username

Your password

Summary

- Create User model with Mongoose
 - Pre-save hook to salt/hash passwords
 - Instance method to compare passwords
- Implement user API to authenticate/signup users
 - Sign JWT tokens with user name
- Add a JWT Strategy to Passport.js
- Use `passport.authenticate(...)` to secure server-side routes
 - Add to middleware stack.